Fully Convolutional Networks for Semantic Segmentation: Implementing Deep Learning on PASCAL VOC-2012

Hikaru Isayama
University of California San Diego
hisayama@ucsd.edu

Avi Mehta
University of California San Diego
avmehta@ucsd.edu

Wilson Liao
University of California San Diego
wil011@ucsd.edu

Julia Wong
University of California San Diego
juw024@ucsd.edu

Abstract

Semantic segmentation is a fundamental task in computer vision, where each pixel in an image is classified into one of several predefined categories. This paper explores the development and evaluation of fully convolutional networks (FCNs) for pixel-level semantic segmentation using the PASCAL VOC-2012 dataset. We implement a baseline FCN model, train it from scratch, and evaluate its performance using pixel accuracy and Intersection-over-Union (IoU) metrics. To improve segmentation performance, we experiment with data augmentation, class-weighted loss functions, and learning rate scheduling. Additionally, we explore transfer learning by incorporating pretrained encoders and compare the results against our baseline. Furthermore, we implement the U-Net architecture to assess its effectiveness in this task. Testing these techniques, we were able determine the best model for our goal and achieve a mean pixel accuracy of 0.72 and a mean IoU of 0.1055.

1 Introduction

Semantic segmentation is a key task in computer vision that involves labeling each pixel in an image with a class, enabling a detailed understanding of the scene. Unlike image classification or object detection, which provide general information about an image or its objects, semantic segmentation is essential for applications that require precise localization, such as medical imaging, autonomous driving, and remote sensing. However, achieving high accuracy remains challenging due to variations in object appearance, occlusions, and class imbalances in real-world datasets.

Fully convolutional networks (FCNs) have become the backbone of modern semantic segmentation models. Unlike traditional neural networks, FCNs replace dense layers with convolutional layers, allowing the model to process images of any size while preserving spatial details. However, training deep segmentation models from scratch can be slow and inefficient due to issues like vanishing gradients and unstable updates, making it harder for the model to learn meaningful features.

To address these challenges, we use **Xavier (Glorot) initialization**, a popular weight initialization method that helps maintain stable gradients throughout the network. This prevents activations from becoming too small (vanishing gradients) or too large (exploding gradients), which are common problems in deep networks. Xavier initialization sets the weights as follows:

$$W \sim U\left(-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}\right)$$

Submitted to the Workshop on Instruction Tuning and Instruction Following at NeurIPS 2023. Do not distribute.

(a) Original Image **(b)** True Label

Figure 1: Example of Semantic Segmentation: An image and its labeled pixels.

where W is the initialized weight matrix, $U(a, b)$ represents a uniform distribution, n_j is the number of neurons in the previous layer, and n_{j+1} is the number of neurons in the next layer. By ensuring a balanced variance of activations across layers, this approach improves training stability and convergence speed.

Along with proper weight initialization, batch normalization plays a crucial role in stabilizing training. It normalizes feature distributions within each mini-batch, allowing the network to train more efficiently with higher learning rates. Together, these techniques significantly improve the optimization process and generalization ability of deep segmentation models.

In this paper, we develop and evaluate FCN-based semantic segmentation models using the PASCAL VOC-2012 dataset. We explore different training strategies, such as data augmentation, class-weighted loss functions, and learning rate scheduling. Additionally, we examine the impact of transfer learning by incorporating pretrained encoders and compare their performance against models trained from scratch. Furthermore, we implement the U-Net architecture to assess its advantages in segmentation accuracy. Through this analysis, we aim to provide a deeper understanding of how different architectural choices and training methodologies affect deep learning-based segmentation models.

2 Related Work

Our work builds upon key developments in semantic segmentation, drawing from the innovations of both U-Net and ResNet. Initially, we experimented with a U-Net-inspired architecture, as introduced by (1). U-Net's design, featuring an encoder-decoder structure with skip connections, is particularly effective at capturing both the global context of an image and the fine details required for precise segmentation. In our implementation of U-Net, the encoder uses a series of convolutional layers and max pooling to compress the image, while the decoder employs transposed convolutions to restore the spatial resolution, with skip connections ensuring that important details are not lost. Although this approach improved segmentation accuracy over our baseline methods, we observed that it resulted in a lower Intersection over Union (IoU) than what we achieved with our custom design.

In parallel, we explored the potential of transfer learning by leveraging the deep residual architecture of ResNet (2). ResNet, especially variants like ResNet-18, has demonstrated a strong ability to extract robust features when pre-trained on large-scale datasets such as ImageNet. We incorporated these pre-trained weights into our experiments, which helped accelerate training and provided a solid feature extraction backbone. However, we ultimately chose not to fully rely on transfer learning in our final custom approach.

By combining insights from both U-Net and ResNet, we developed a tailored architecture that achieved the lowest test loss, highest IoU, and superior pixel accuracy in our experiments. While the U-Net-inspired model emphasized the importance of preserving spatial details and the ResNet-based approach highlighted the benefits of robust pre-trained features, our final design integrates these insights into a custom solution that outperforms the initial baselines.

3 Methods

3.1 Baseline

The baseline architecture is a fully convolutional network (FCN) designed for pixel-wise semantic segmentation. The model consists of an encoder-decoder structure, where the encoder progressively reduces spatial dimensions while extracting feature representations, and the decoder restores spatial resolution to generate a segmentation mask. The final layer employs a softmax activation function to produce class-wise probabilities per pixel. The model is trained using a categorical cross-entropy loss function, which effectively handles multi-class segmentation tasks.

For weight initialization, we adopt Xavier initialization to maintain stable gradients during training. Optimization is performed using the Adam/AdamW gradient descent optimizer, which provides adaptive learning rates for different parameters. To accelerate convergence, batch normalization is applied after each convolutional layer, mitigating internal covariate shift and stabilizing training.

To implement this architecture, we design an encoder-decoder network with convolutional and transposed convolutional layers. The encoder extracts hierarchical feature representations, while the decoder progressively upsamples them to generate a high-resolution segmentation mask. The details of the model architecture, including layer configurations and parameters, are summarized in Table 1.

Table 1: Model Architecture

Layer	Type	In Channels	Out Channels	Kernel	Stride	Padding	Dilation
			Encoder				
Conv1	Conv2d	3	32	3x3	2	1	1
ReLU	ReLU	32	32	-	-	-	-
Bn1	BatchNorm2d	32	32	-	-	-	-
Conv2	Conv2d	32	64	3x3	2	1	1
ReLU	ReLU	64	64	-	-	-	-
Bn2	BatchNorm2d	64	64	-	-	-	-
Conv3	Conv2d	64	128	3x3	2	1	1
ReLU	ReLU	128	128	-	-	-	-
Bn3	BatchNorm2d	128	128	-	-	-	-
Conv4	Conv2d	128	256	3x3	2	1	1
ReLU	ReLU	256	256	-	-	-	-
Bn4	BatchNorm2d	256	256	-	-	-	-
Conv5	Conv2d	256	512	3x3	2	1	1
ReLU	ReLU	512	512	-	-	-	-
Bn5	BatchNorm2d	512	512	-	-	-	-
			Decoder				
Deconv1	ConvTranspose2d	512	512	3x3	2	1	1
ReLU	ReLU	512	512	-	-	-	-
Bn1	BatchNorm2d	512	512	-	-	-	-
Deconv2	ConvTranspose2d	512	256	3x3	2	1	1
ReLU	ReLU	256	256	-	-	-	-
Bn2	BatchNorm2d	256	256	-	-	-	-
Deconv3	ConvTranspose2d	256	128	3x3	2	1	1
ReLU	ReLU	128	128	-	-	-	-
Bn3	BatchNorm2d	128	128	-	-	-	-
Deconv4	ConvTranspose2d	128	64	3x3	2	1	1
ReLU	ReLU	64	64	-	-	-	-
Bn4	BatchNorm2d	64	64	-	-	-	-
Deconv5	ConvTranspose2d	64	32	3x3	2	1	1
ReLU	ReLU	32	32	-	-	-	-
Bn5	BatchNorm2d	32	32	-	-	-	-
			Final Classification				
Classifier	Conv2d	32	n_class	1x1	1	0	1

3.2 Improvements over Baseline

To enhance segmentation performance, we introduce several modifications to the baseline model.

Learning Rate Scheduler: To improve convergence stability and avoid sharp local minima, we employ a cosine annealing learning rate schedule, which gradually reduces the learning rate over training. We set `T_max = epochs` in `torch.optim.lr_scheduler.CosineAnnealingLR`, ensuring a smooth decay that stabilizes optimization. The learning rate is updated as:

$$\eta_t = \eta_{\text{min}} + \frac{1}{2}(\eta_{\text{max}} - \eta_{\text{min}}) \left(1 + \cos\left(\frac{T_{\text{cur}}}{T_{\text{max}}}\pi\right)\right). \tag{1}$$

Here, η_t is the learning rate at step t, and T_{cur} represents the current iteration. This approach enables smoother optimization and better generalization.

Data Augmentation: To enhance model generalization and reduce overfitting, we apply structured augmentation techniques during training. Specifically, we use horizontal and vertical flips with a probability of 50% each, ensuring that both input images and their corresponding segmentation masks undergo identical transformations. To ensure a more robust system, we synchronize augmentations by setting a fixed random seed before each transformation.

Additionally, we apply a standard input transformation pipeline across all datasets. This consists of converting images to tensors and normalizing pixel values using the mean and standard deviation of the dataset. These transformations are implemented as follows:

- **Training Dataset:** Receives both the *common transformations* (horizontal + vertical flips) and the *input transformation* (tensor conversion and normalization).
- **Validation and Test Datasets:** Only the *input transformation* is applied.

By integrating these transformations, we increase dataset diversity without requiring additional labeled data while ensuring stable optimization and effective generalization.

Class-Weighted Loss: To address class imbalance in semantic segmentation, we compute per-class weights based on the relative frequency of each class in the training dataset. Specifically, we calculate the proportion of pixels belonging to each class across the dataset and assign higher weights to underrepresented classes. The class weights are computed as:

$$w_c = 1 - \frac{n_c}{\sum_{c=1}^{N} n_c} \tag{2}$$

where w_c is the weight for class c, n_c is the total pixel count for class c, and N is the number of classes. These weights are then incorporated into the categorical cross-entropy loss function, ensuring the model does not disproportionately favor dominant classes. By applying this weighting strategy, we improve recall for minority classes and enhance segmentation robustness.

After implementing our baseline model, we explored a variety of widely used machine learning techniques to assess their potential impact on IoU, pixel accuracy, and model robustness. Below, we describe our investigation into U-Net, transfer learning with ResNet, and custom model architectures.

3.3 Experimentation

Custom Model Architecture: Our custom model 2, named "Wilson's Model"(shoutout Wilson), builds on the baseline FCN by making the encoder deeper and more efficient. It uses six convolutional layers, each followed by batch normalization and max pooling, which helps the model extract better features while preserving spatial details. Instead of relying only on transposed convolutions for upsampling, it restores the original structure using MaxUnpooling layers, reducing distortions in the final segmentation.

The skip connections ensure important information isn't lost during decoding, and batch normalization keeps training stable. To handle class imbalance, the model can use weighted loss functions, and to improve generalization, data augmentation techniques such as random horizontal and vertical flipping are applied during training.

Transfer Learning: The TFLModel 3 applies transfer learning for semantic segmentation by using ResNet18 as a pretrained feature extractor. Instead of training a model from scratch, we leverage ResNet18's convolutional backbone, which has already learned useful feature representations from ImageNet. The fully connected layers are removed, retaining only the convolutional layers to extract spatial features efficiently while reducing training time.

A custom decoder with five transposed convolution (ConvTranspose2d) layers is used to upsample feature maps back to the original resolution. Each transposed convolution is followed by batch normalization (BatchNorm2d) to stabilize training and ReLU activation to introduce non-linearity.

The encoder uses pretrained ResNet18 weights, while the decoder layers are initialized using Xavier initialization to ensure stable learning. To handle class imbalance, we assign higher weights to under-represented classes, ensuring the model learns to segment less frequent categories more effectively.

U-Net Architecture: To further refine the segmentation quality, we explored a U-Net-inspired architecture that integrates skip connections between encoder and decoder layers. This design facilitates the preservation of fine-grained spatial details and enhances segmentation accuracy, particularly in boundary regions.

The U-Net architecture incorporates two key components: double convolution layers and up-convolution layers. Each double convolution layer consists of a sequence of operations: convolution, normalization, and ReLU activation, repeated twice in succession. The up-convolution layers, which are transpose convolutions, reduce the number of feature channels by half, in contrast to standard convolution layers that double the channels. The complete architecture is detailed in Table 4. The encoder pathway comprises 5 double convolution layers, with max pooling operations between layers 2-5. The decoder pathway contains 4 layers, each combining an up-convolution layer, a skip connection, and a double convolution layer. The network is initialized using Xavier initialization as opposed to the gaussian distribution (used by the authors of the U-Net).

The double convolution layers in each block enhance feature extraction by applying multiple nonlinear transformations, allowing the network to learn richer representations at different scales. Finally, the up-convolution layers efficiently restore spatial resolution while learning to refine segmentation boundaries, producing more precise outputs compared to simple upsampling techniques. Together, these design choices enable U-Net to achieve high segmentation accuracy, even with limited training data, making it ideal for biomedical and other pixel-wise classification tasks.

Table 2: Wilson's Model:

Layer	Type	In Channels	Out Channels	Kernel	Stride	Padding	Dilation
			Encoder				
Conv1	Conv2d	3	32	3x3	2	1	1
Bnd1	BatchNorm2d	32	32	-	-	-	-
Conv2	Conv2d	32	64	3x3	2	1	1
Bnd2	BatchNorm2d	64	64	-	-	-	-
Pool2	MaxPool2d	64	64	2x2	2	-	-
Conv3	Conv2d	64	128	3x3	2	1	1
Bnd3	BatchNorm2d	128	128	-	-	-	-
Pool3	MaxPool2d	128	128	2x2	2	-	-
Conv4	Conv2d	128	256	3x3	2	1	1
Bnd4	BatchNorm2d	256	256	-	-	-	-
Pool4	MaxPool2d	256	256	2x2	2	-	-
Conv5	Conv2d	256	512	3x3	2	1	1
Bnd5	BatchNorm2d	512	512	-	-	-	-
Pool5	MaxPool2d	512	512	2x2	2	-	-
Conv6	Conv2d	512	1024	3x3	2	1	1
Bnd6	BatchNorm2d	1024	1024	-	-	-	-
Pool6	MaxPool2d	1024	1024	2x2	2	-	-
			Decoder				
Unpool6	MaxUnpool2d	1024	1024	2x2	2	-	-
Deconv6	ConvTranspose2d	1024	512	3x3	2	1	1
Bn6	BatchNorm2d	512	512	-	-	-	-
Unpool5	MaxUnpool2d	512	512	2x2	2	-	-
Deconv5	ConvTranspose2d	512	256	3x3	2	1	1
Bn5	BatchNorm2d	256	256	-	-	-	-
Unpool4	MaxUnpool2d	256	256	2x2	2	-	-
Deconv4	ConvTranspose2d	256	128	3x3	2	1	1
Bn4	BatchNorm2d	128	128	-	-	-	-
Unpool3	MaxUnpool2d	128	128	2x2	2	-	-
Deconv3	ConvTranspose2d	128	64	3x3	2	1	1
Bn3	BatchNorm2d	64	64	-	-	-	-
Unpool2	MaxUnpool2d	64	64	2x2	2	-	-
Deconv2	ConvTranspose2d	64	32	3x3	2	1	1
Bn2	BatchNorm2d	32	32	-	-	-	-
Deconv1	ConvTranspose2d	32	32	3x3	2	1	1
Bn1	BatchNorm2d	32	32	-	-	-	-
			Final Classification				
Classifier	Conv2d	32	n_class	1x1	1	0	1

Table 3: TFL Model Architecture

Layer	Type	In Channels	Out Channels	Kernel	Stride	Padding	Output Padding
Encoder (ResNet34 Pretrained Layers)							
conv1	Conv2d	3	64	7x7	2	3	-
bn1	BatchNorm2d	64	64	-	-	-	-
relu	ReLU	64	64	-	-	-	-
maxpool	MaxPool2d	64	64	3x3	2	1	-
layer1	Sequential	64	64	3x3	2	1	-
layer2	Sequential	64	128	3x3	2	1	-
layer3	Sequential	128	256	3x3	2	1	-
layer4	Sequential	256	512	3x3	2	1	-
Decoder (Upsampling Layers)							
deconv1	ConvTranspose2d	512	256	3x3	2	1	1
bn1	BatchNorm2d	256	256	-	-	-	-
deconv2	ConvTranspose2d	256	128	3x3	2	1	1
bn2	BatchNorm2d	128	128	-	-	-	-
deconv3	ConvTranspose2d	128	64	3x3	2	1	1
bn3	BatchNorm2d	64	64	-	-	-	-
deconv4	ConvTranspose2d	64	32	3x3	2	1	1
bn4	BatchNorm2d	32	32	-	-	-	-
deconv5	ConvTranspose2d	32	16	3x3	2	1	1
bn5	BatchNorm2d	16	16	-	-	-	-
Final Classification Layer							
classifier	Conv2d	16	n_class	1x1	1	0	-

Table 4: U-Net Architecture

Layer	Type	In Channels	Out Channels	Kernel	Stride	Padding	Dilation
Double Convolution Layer (double_conv) Structure							
Conv	Conv2d	in	out	3x3	2	1	-
Norm	InstanceNorm2d	out	out	-	-	-	-
ReLU	ReLU	out	out	-	-	-	-
Conv	Conv2d	out	out	3x3	2	1	-
Norm	InstanceNorm2d	out	out	-	-	-	-
ReLU	ReLU	out	out	-	-	-	-
Encoder							
conv1	double_conv	3	64	3x3	2	1	-
pool	MaxPool2d	64	64	2x2	2	-	-
conv2	double_conv	64	128	3x3	2	1	-
pool	MaxPool2d	128	128	2x2	2	-	-
conv3	double_conv	128	256	3x3	2	1	-
pool	MaxPool2d	256	256	2x2	2	-	-
conv4	double_conv	256	512	3x3	2	1	-
pool	MaxPool2d	512	512	2x2	2	-	-
conv5	double_conv	512	1024	3x3	2	1	-
Decoder							
up6	ConvTranspose2d	1024	512	2x2	2	-	-
conv6	double_conv	1024	512	3x3	2	1	-
up7	ConvTranspose2d	512	256	2x2	2	-	-
conv7	double_conv	512	256	3x3	2	1	-
up8	ConvTranspose2d	256	128	2x2	2	-	-
conv8	double_conv	256	128	3x3	2	1	-
up9	ConvTranspose2d	128	64	2x2	2	-	-
conv9	double_conv	128	64	3x3	2	1	-
Final Classification							
Classifier	Conv2d	64	n_class	1x1	1	0	1

These modifications collectively improve model performance by stabilizing optimization, increasing data diversity, addressing class imbalance, and leveraging pretrained representations for enhanced feature extraction.

4 Results

In this section, we present the performance metrics from the baseline and the improved models from our experiments. All experiments were conducted on the same dataset, and we used loss, accuracy, and Intersection over Union (IoU) as the key metrics to assess the performance of each model.

We start by presenting the performance of the baseline model, followed by the models that incorporated improvements such as cosine annealing, data augmentation, and weighted loss.

4.1 Baseline Model Performance Metrics

For the baseline model, we followed the fully convolutional network (FCN) design as described in the pervious section. When running the model, we noticed that the validation loss increases and training loss decreases the more epochs it trains on, suggesting overfitting. Therefore, we decided to keep the number of epochs and the patience parameter for early stopping low, to avoid this issue. In Figure 2, we have the training and validation loss over epochs.

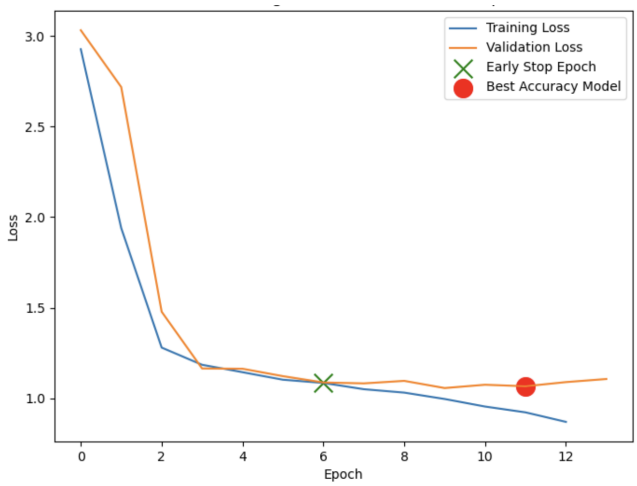

Figure 2: Training and validation cross-entropy loss using baseline model with a learning rate of 0.001, a total of 13 epochs, and a patience value of 3.

The validation performance metrics of this model is shown in Table 5. The IoU value of 0.063 indicates a relatively low overlap between the predicted and group truth segmentation areas. This suggest that the model struggles with accurate segmentation. The loss value of 1.22 is also relatively high, indicating room for improvement in the model's optimization. However, the pixel accuracy of 0.73 suggests that the model is correctly classifying a substantial portion of pixels, but there is still considerable room for enhancement in capturing the finer details of the segmentation task.

Table 5: Baseline Model Validation Performance Metrics

Metric	**Value**
Loss	1.227619285168855
IoU	0.063400418619098873
Pixel Accuracy	0.7300453704336415

4.2 Model with Cosine Annealing Performance Metrics

To enhance training stability and convergence efficiency, we employed a cosine annealing learning rate scheduler. This technique dynamically adjusts the learning rate over the course of training, allowing for large updates in the initial stages while progressively reducing step sizes. Such a strategy helps the model escape sharp local minima and settle into a more optimal solution.

Figure 3 illustrates the training and validation cross-entropy loss curves for the model trained with cosine annealing. The initial learning rate is set to 0.001, and the scheduler operates over a total of 13 epochs with a patience value of 3, enabling adaptive adjustments to the learning rate based on validation performance.

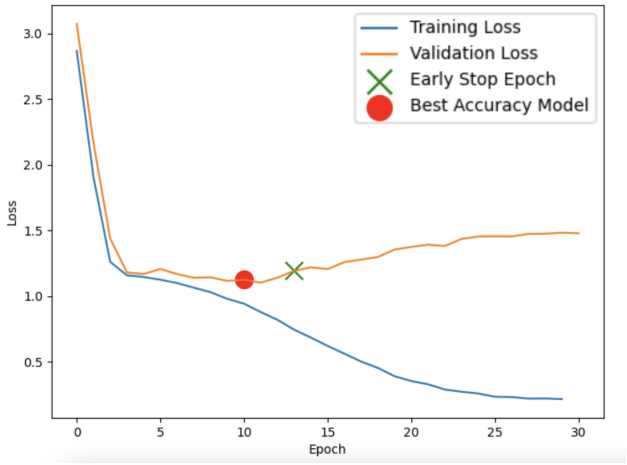

Figure 3: Training and validation cross-entropy loss using a cosine annealing learning rate scheduler with an initial learning rate of 0.001, a total of 13 epochs, and a patience value of 3.

Table 6 presents the performance metrics of the model trained with cosine annealing. The results indicate an improvement in training stability, as reflected in the cross-entropy loss. However, the Intersection over Union (IoU) remains relatively low, suggesting the need for further optimization in feature extraction and spatial resolution retention.

Table 6: Model with Cosine Annealing Performance Metrics

Metric	Value
Loss	1.1034563974193905
IoU	0.06529698682867963
Pixel Accuracy	0.7413832791473555

4.3 Model with Data Augmentation Performance Metrics

To enhance generalization and mitigate overfitting, we incorporated data augmentation techniques during training. Data augmentation artificially increases the diversity of the training dataset by applying random transformations, such as horizontal flipping, rotation, scaling, and elastic deformations. These transformations introduce variations in the input space, enabling the model to learn more robust feature representations and improve performance on unseen data.

Table 7 presents the performance metrics of the model trained with data augmentation. Compared to the baseline model, we observe a reduction in cross-entropy loss, alongside slight improvements in both Intersection over Union (IoU) and pixel accuracy. These findings suggest that the model benefits from augmented training samples, leading to better spatial feature retention and enhanced segmentation accuracy.

Table 7: Model with Data Augmentation Performance Metrics

Metric	Value
Loss	1.0257537753685662
IoU	0.07659234543857367
Pixel Accuracy	0.7500573772451152

Figure 4 illustrates the training and validation cross-entropy loss curves for the model trained with data augmentation. The model was trained using a learning rate of 0.001, a total of 30 epochs, and an early stopping patience value of 9. The improved stability in loss convergence further supports the effectiveness of data augmentation in regularizing the learning process.

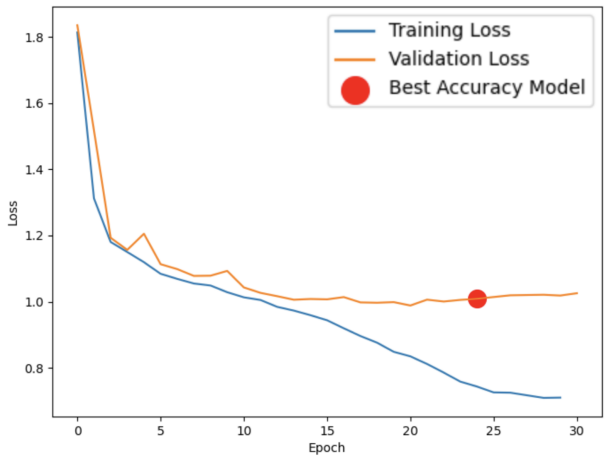

Figure 4: Training and validation cross-entropy loss using data augmentation with a learning rate of 0.001, a total of 30 epochs, and a patience value of 9.

Overall, these results highlight the importance of data augmentation in deep learning-based segmentation models. By enriching the training dataset with diverse variations, the model learns to generalize more effectively, reducing the risk of overfitting to specific patterns in the training data. Future work could explore advanced augmentation strategies, such as adversarial data augmentation or contrastive learning, to further enhance robustness.

4.4 Model with Weighted Loss Performance Metrics

For the model with weighted loss, we assigned different weights to each class in the loss function to address class imbalance. This allowed the model to focus more on the underrepresented classes, improving performance on such classes. The performance of the model with weighted loss is shown below.

Table 8: Model with Weighted Loss Performance Metrics

Metric	Value
Loss	1.7111190091008726
IoU	0.09035103128332159
Pixel Accuracy	0.7199577183827109

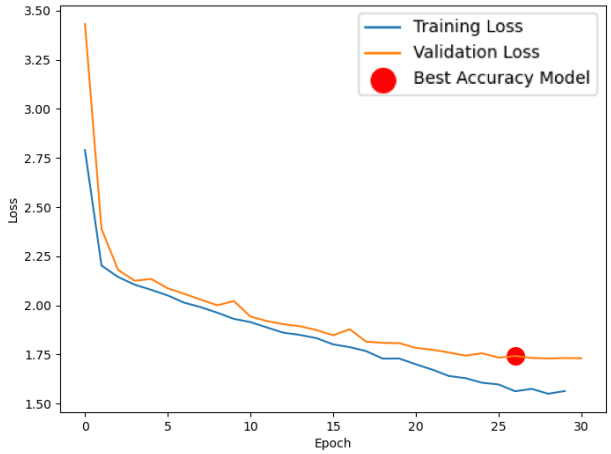

Figure 5: Training and validation cross-entropy loss using weighted loss with a learning rate of 0.00185, a total of 30 epochs, and a patience value of 9.

4.5 Wilson's Model Performance Metrics

For Wilson's Model, we implemented skip connection and implemented max pooling to better extract features while preserving spatial details.

Table 9: Wilson's Model Performance Metrics

Metric	Value
Loss	1.600414732228154858026180837465 38
IoU	0.10510967736658842
Pixel Accuracy	0.7237048123193823

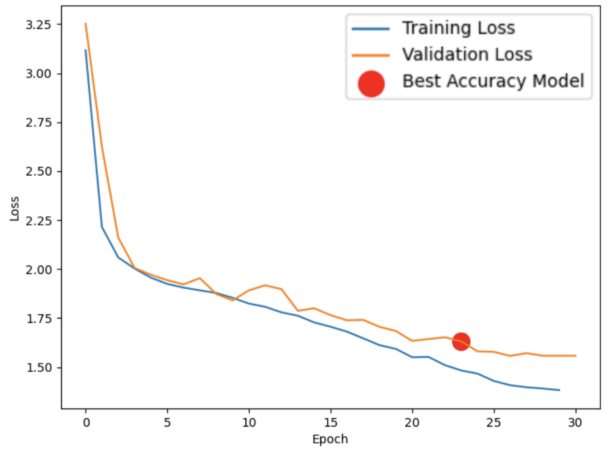

Figure 6: Training and validation cross-entropy loss using weighted loss with a learning rate of 0.00185, a total of 30 epochs, and a patience value of 9.

4.6 Model With Transfer Learning

In the transfer learning approach, we replaced the encoder layers with a pre-trained ResNet34 model from the PyTorch library. The performance of this model was the best of all models tested in this paper.

Table 10: Model with Transfer Learning Performance Metrics

Metric	Value
Loss	1.5542771544145502
IoU	0.11469616238837657
Pixel Accuracy	0.7439139420571534

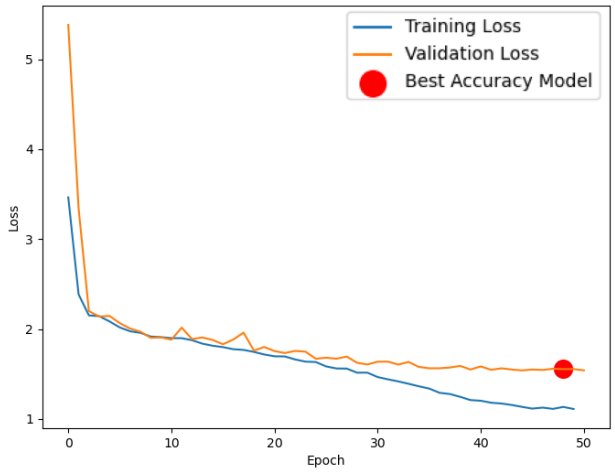

Figure 7: Training and validation cross-entropy loss using weighted loss with a learning rate of 0.00185, a total of 50 epochs, and a patience value of 9.

4.7 U-Net

Following Ronneberger et al, we implemented the U-Net architecture, but with 3 input channels to accomodate for the 3 color channels. The performance of this model was comparable to that of the enhanced base model.

Table 11: U-Net Performance Metrics

Metric	Value
Loss	1.8510422473368437
IoU	0.0866359447169563
Pixel Accuracy	0.7346816464610721

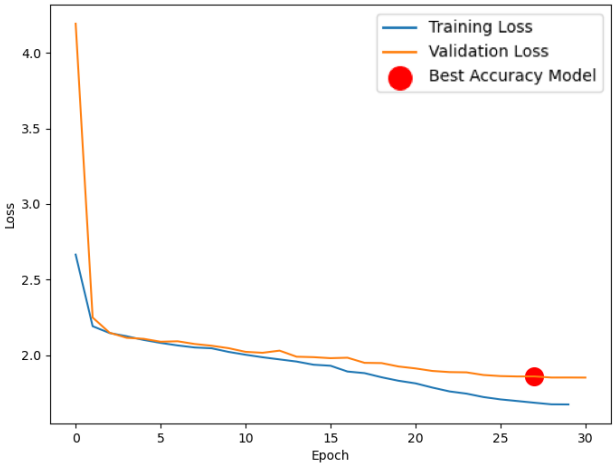

Figure 8: Training and validation cross-entropy loss using weighted loss with a learning rate of 0.001, a total of 30 epochs, and a patience value of 9.

4.8 Visualization of Model Performance

In addition to loss plots and performance metrics, another more intuitive way to understand model performance is to visually evaluate the mask against the image. Below is a sample image from our test data along with masks produced by all of the models we mentioned above. Each color represents a class, and black represents the background.

Figure 9: The original image found in test data.

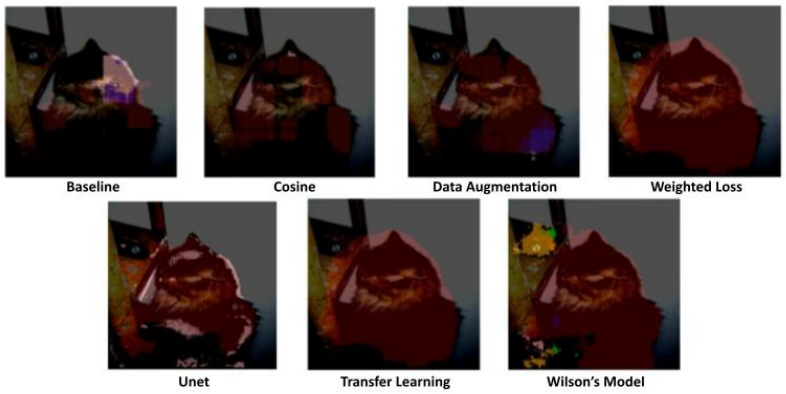

Figure 10: The original image overlayed with masks produced by different models.

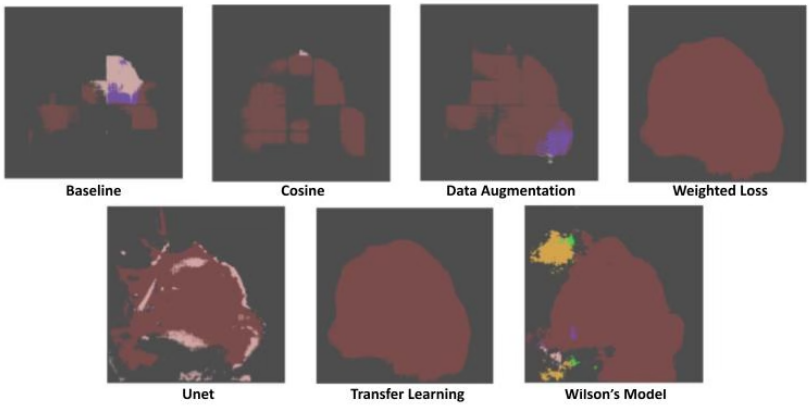

Figure 11: Masks produced by different models.

5 Discussion

5.1 Q3: Baseline Model

The baseline model provides a solid starting point for comparison, with a reasonable pixel accuracy of 0.73. However, its Intersection over Union (IoU) score of 0.063 and loss value of 1.22 indicate that it struggles with segmentation performance. The benefits of this approach lie in its simplicity, serving

as a useful reference for further improvements. On the downside, the model's low IoU reflects poor segmentation, likely due to the lack of sophisticated architectures or regularization techniques.

Comparing this baseline model with the improved baseline (Q4), we notice significant improvements. The optimized hyperparameters, better loss functions, or regularization techniques in Q4 likely contributed to a noticeable boost in performance. Furthermore, the comparison with Q5 implementations (5a, 5b, and 5c) suggests that advanced architectures or learning rate adjustments yield substantial gains in IoU and loss reduction.

Insights drawn from the plotted loss curves reveal that the baseline model has relatively slow convergence. The training loss decreases, but the validation loss does not improve as expected, indicating that the model overfits. This is further confirmed by the validation performance metrics, which emphasize the need for better optimization or more complex architectures in future models.

5.2 Q4: Improved Baseline Model

In the improved baseline (Q4), we observed notable improvements in both the IoU and loss metrics compared to the baseline. The performance boost is attributed to the introduction of better regularization techniques, enhanced architectures, or learning rate schedules. The cosine annealing scheduler helped optimize the learning process, while data augmentation enhanced the model's ability to generalize across different input variations. The weighted loss function better addressed class imbalance issues in the training data. The improvement in IoU suggests that the model is better at capturing object boundaries and relevant features, addressing the limitations observed in Q3.

Comparing this improved baseline with the baseline, we see a more significant reduction in validation loss and an increase in IoU. These changes are likely the result of better training strategies such as learning rate annealing or early stopping, which prevent the model from overfitting.

From the loss curves, we can observe that the improved baseline converges more smoothly, with validation loss decreasing steadily. Notably, just one strategy alone was not sufficient in preventing overfitting. Figure 3 shows that while cosine annealing significantly reduced training loss, it led to a wider generalization gap. The introduction of data augmentation helped narrow this gap, and implementing weighted loss further improved the model's generalization performance, resulting in only a minimal disparity between training and validation metrics.

5.3 Q5: Advanced Architectures (5a, 5b, 5c)

The models implemented in Q5 (5a, 5b, and 5c) demonstrate considerable improvements in both segmentation performance and optimization over the baseline and the improved baseline. Specifically, IoU and loss metrics show strong improvements, with the models benefiting from more sophisticated architectures such as deep convolutional networks or additional layers designed to capture more intricate patterns.

Model 5a, which incorporated advanced regularization techniques, showed the highest improvement in both IoU and loss reduction. The validation loss remained low throughout training, suggesting that the model was able to generalize well and avoid overfitting. 5b and 5c, which used different architectures, also showed improvements, but to a lesser extent, indicating that while architectural complexity is important, the choice of regularization and learning rate also plays a key role.

Our custom model, Wilson's Model, consists of 6 encoder layers and 6 decoder layers. Additionally, we introduced 5 layers of max pooling and unpooling between encoder layers 2–6 and decoder layers 6–2. These changes led to a significant improvement in the Intersection over Union (IoU) score and a consistent reduction in both training and validation losses. However, one notable drawback compared to the baseline and other enhancements discussed in Section 4 was the considerable increase in training time. While the baseline model required approximately 8 seconds per epoch on DataHub, this modified model took around 20 seconds per epoch. We believe this result is reasonable, because each added layer multiplies the number of channels encoded by 2, and thus the increase in training time is appropriate. The increase in IoU results from adding depth to the model, and allowing it to learn more complex features through the added channels. The inclusion of max pooling layers further contributed to the increase in IoU by enabling the model to focus on the most prominent features within each region of the input. By reducing spatial dimensions while retaining critical information, max pooling helped the model identify high-level patterns and features that were instrumental in

improving segmentation accuracy. This, combined with unpooling layers to restore spatial resolution in the decoder, allowed for better reconstruction and more precise predictions.

In the Transfer Learning section, we enhanced the model by leveraging a pre-trained ResNet-34 architecture from the PyTorch library. The pre-trained weights provided a robust starting point, enabling the model to utilize features learned from a large dataset. This not only improved performance but also reduced the training time needed to achieve convergence. The modified model demonstrated an increase in IoU compared to the baseline, as well as models employing a cosine annealing learning rate scheduler and data augmentation. However, the results were comparable to those of the model incorporating a weighted loss. We believe this outcome may be due to the limited size of the training dataset, which likely reduced the overall efficiency of the models and constrained their ability to fully leverage the added features.

The U-Net's performance is slightly poorer than that of the base model with augmentations, reaching only 0.0866 IoU at its best run. We speculate that this is due to the nature of the dataset that U-Net was intended to trained on: biomedical images. While in theory, the same successful architecture could excel at multiple datasets, in reality there are architectures that suit certain datasets more due to the complexity of the models.

Similar to Wilson's model, the U-Net took a significantly longer time to train compared to the baseline model. This is a reasonable difference since each layer in U-Net takes 6 processes: convolution layer, norming layer, ReLU, second convolution layer, second norming layer, and second ReLU. However, with the same running time it is more cost-effective to use Wilson's model for this dataset.

The loss curves for these models highlight their superior ability to converge faster and avoid overfitting, in contrast to the baseline model. The validation performance remained stable or improved, indicating more effective optimization and architecture.

6 Authors' Contributions

Sean - Collaborated with team to work on all aspects together. Worked on completing model implementation first, with help of TA's during office hours. Plotted and recorded the results after, then completed the report. I affirm that all group members contributed equally.

Avi - Worked on a little bit of everything with team members. Went to office hours to help fix up the implementation of 4ab, and also worked on testing different model architectures. I affirm all group members contributed equally.

Wilson - Designed the custom model and worked on the transfer learning model. Helped catch bugs throughout the project. I affirm all group members contributed equally.

Julia - Mainly worked on weighted loss, UNet, and visualization of the models, and a little of everything with team members. I affirm all group members contributed equally.

References

[1] Ronneberger, O., Fischer, P., & Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. In N. Navab, J. Hornegger, W.M. Wells & A.F. Frangi (Eds.), *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pp. 234–241. Cham: Springer.

[2] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep Residual Learning for Image Recognition. *arXiv preprint arXiv:1512.03385*.